Applications of Automata and Concurrency Theory in Networks

Alexandra Silva

University College London

CONCUR 2015
Context

Automata

Coalgebra

Concurrency Software-defined Networks
Automata

- Automata are basic structures in Computer Science
- Language equivalence: well-studied, several algorithms
Automata are basic structures in Computer Science
Language equivalence: well-studied, several algorithms

New perspectives ↩️ key algorithmic improvements
Concurrent Theory: labelled transition systems

- Central research topic: a spectrum of equivalences
The spectrum of equivalences

bisimulation semantics

2-nested simulation semantics

ready simulation semantics

possible-worlds semantics

possible-futures semantics

ready trace semantics

failure trace semantics

readiness semantics

simulation semantics

failures semantics

completed trace semantics

trace semantics
Many efficient algorithms for equivalence of automata.
Applications in concurrency?
Many efficient algorithms for equivalence of automata.

Applications in concurrency?

Various spectrum equivalences

= Language equivalence of a *transformed* system

= Automaton with outputs and structured states (Moore automaton).

Bonsangue, Bonchi, Caltais, Rutten, Silva. MFPS 12
Generalization of existing algorithms to Moore automata
Generalization of existing algorithms to Moore automata

Small step conceptually, great impact application-wise

Method: coalgebra

Bonchi, Caltais, Pous, Silva. APLAS 2013
Algorithm derivation from the type

Equivalence/Minimization algorithms from the type $X \to TX$?

<table>
<thead>
<tr>
<th>Type</th>
<th>Equation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Deterministic automata</td>
<td>$X \to 2 \times X^A$</td>
</tr>
<tr>
<td>Moore automata</td>
<td>$X \to B \times X^A$</td>
</tr>
<tr>
<td>Linear weighted automata</td>
<td>$V \to \mathbb{R} \times V^A$</td>
</tr>
<tr>
<td>KAT automata</td>
<td>$B \to B \times B^{B \times A}$</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Algorithm derivation from the type

Equivalence/Minimization algorithms from the type $X \rightarrow T X$?

Deterministic automata $X \rightarrow 2 \times X^A$

Moore automata $X \rightarrow B \times X^A$

Linear weighted automata $V \rightarrow \mathbb{R} \times V^A$

KAT automata $B \rightarrow \mathbb{B} \times B^{B \times A}$
Networking

“The last bastion of mainframe computing” [Hamilton 2009]

- Modern computers
  - implemented with commodity hardware
  - programmed using general-purpose languages, standard interfaces

- Networks
  - built and programmed the same way since the 1970s
  - low-level, special-purpose devices implemented on custom hardware
  - routers and switches that do little besides maintaining routing tables and forwarding packets
  - configured locally using proprietary interfaces
  - network configuration ("tuning") largely a black art
Networking

- Difficult to implement end-to-end routing policies and optimizations that require a global perspective
- Difficult to extend with new functionality
- Effectively impossible to reason precisely about behavior
Main idea behind SDN

A general-purpose controller manages a collection of programmable switches

- controller can monitor and respond to network events
  - new connections from hosts
  - topology changes
  - shifts in traffic load
- controller can reprogram the switches on the fly
  - adjust routing tables
  - change packet filtering policies
SDN Network Architecture

Your Program goes here!

Ox Controller Platform

or POX, Beacon, Floodlight, others

OpenFlow API

OpenFlow-compatible switches
Pica8, Dell, NEC, HP, many others
Software Defined Networks (SDN)

Controller has a global view of the network

Enables a wide variety of applications:

- standard applications
  - shortest-path routing
  - traffic monitoring
  - access control
- more sophisticated applications
  - load balancing
  - intrusion detection
  - fault tolerance
Software Defined Networks (SDN)

“In the SDN architecture, the control and data planes are **decoupled**, network intelligence and state are **logically centralized**, and the underlying network infrastructure is **abstracted from the applications**. As a result, enterprises and carriers gain unprecedented programmability, automation, and network control, enabling them to build **highly scalable, flexible networks** that readily adapt to changing business needs.”

OpenFlow

A first step: the OpenFlow API [McKeown & al., SIGCOMM 08]

- specifies capabilities and behavior of switch hardware
- a language for manipulating network configurations
- very low-level: easy for hardware to implement, difficult for humans to write and reason about

But... 

- is platform independent
- provides an open standard that any vendor can implement
A Major Trend in Industry

Backbone network runs OpenFlow

Bought by VMware for $1.2B
Network Programming Languages & Analysis Tools

Goals:

- raise the level of abstraction above hardware-based APIs (OpenFlow)
- make it easier to build sophisticated and reliable SDN applications and reason about them
Network Programming Languages & Analysis Tools

Goals:

- raise the level of abstraction above hardware-based APIs (OpenFlow)
- make it easier to build sophisticated and reliable SDN applications and reason about them

- Formally Verifiable Networking [Wang & al., HotNets 09]
- FlowChecker [Al-Shaer & Saeed Al-Haj, SafeConfig 10]
- Anteater [Mai & al., SIGCOMM 11]
- Nettle [Voellmy & Hudak, PADL 11]
- Header Space Analysis [Kazemian & al., NSDI 12]
- Frenetic [Foster & al., ICFP 11] [Reitblatt & al., SIGCOMM 12]
- NetCore [Guha & al., PLDI 13] [Monsanto & al., POPL 12]
- Pyretic [Monsanto & al., NSDI 13]
- VeriFlow [Khurshid & al., NSDI 13]
- Participatory networking [Ferguson & al., SIGCOMM 13]
- Maple [Voellmy & al., SIGCOMM 13]
Network Programming Languages & Analysis Tools

Goals:

- raise the level of abstraction above hardware-based APIs (OpenFlow)
- make it easier to build sophisticated and reliable SDN applications and reason about them

- Formally Verifiable Networking [Wang & al., HotNets 09]
- FlowChecker [Al-Shaer & Saeed Al-Haj, SafeConfig 10]
- Anteater [Mai & al., SIGCOMM 11]
- Nettle [Voellmy & Hudak, PADL 11]
- Header Space Analysis [Kazemian & al., NSDI 12]

▶ Frenetic [Foster & al., ICFP 11] [Reitblatt & al., SIGCOMM 12]
▶ NetCore [Guha & al., PLDI 13] [Monsanto & al., POPL 12]
- Pyretec [Monsanto & al., NSDI 13]
- VeriFlow [Khurshid & al., NSDI 13]
- Participatory networking [Ferguson & al., SIGCOMM 13]
- Maple [Voellmy & al., SIGCOMM 13]
NetKAT papers


Nate Foster, Dexter Kozen, Matthew Milano, Alexandra Silva, and Laure Thompson, A Coalgebraic Decision Procedure for NetKAT. POPL 15.
NetKAT

Simple programming language/logic, expressive enough for basic properties.

Reachability

- Can host A communicate with host B? Can every host communicate with every other host?

Security

- Does all untrusted traffic pass through the intrusion detection system located at C?
- Are non-SSH packets forwarded? Are SSH packets dropped?

Loop detection

- Is it possible for a packet to be forwarded around a cycle in the network?

Policy equivalence

- Given the network topology, are policies $p$ and $q$ equivalent?
NetKAT [Anderson & al. 14]

NetKAT

= Kleene algebra with tests (KAT)
  + additional specialized constructs particular to network topology and packet switching
Kleene Algebra (KA)


\[(0 + 1(01*0)*1)^*\]
\[\{\text{multiples of 3 in binary}\}\]

\[(ab)^* a = a(ba)^*\]
\[\{a, aba, ababa, \ldots\}\]

\[(a + b)^* = a^*(ba^*)^*\]
\[\{\text{all strings over} \ \{a, b\}\}\]

Diagram 1:

Diagram 2:
Foundations of the Algebraic Theory


John Horton Conway
(1937–)
Axioms of KA

Idempotent Semiring Axioms

\[
\begin{align*}
  p + (q + r) &= (p + q) + r & p(qr) &= (pq)r \\
  p + q &= q + p & 1p &= p1 = p \\
  p + 0 &= p & p0 &= 0p = 0 \\
  p + p &= p &  \\
  p(q + r) &= pq + pr & a \leq b & \iff a + b = b \\
  (p + q)r &= pr + qr &  \\
\end{align*}
\]

Axioms for $*$

\[
\begin{align*}
  1 + pp^* &\leq p^* & q + px &\leq x \implies p^*q \leq x \\
  1 + p^*p &\leq p^* & q + xp &\leq x \implies qp^* \leq x \\
\end{align*}
\]
Regular sets of strings over $\Sigma$

\[
A + B = A \cup B \\
AB = \{xy \mid x \in A, \ y \in B\} \\
A^* = \bigcup_{n \geq 0} A^n = A^0 \cup A^1 \cup A^2 \cup \ldots \\
1 = \{\varepsilon\} \\
0 = \emptyset
\]

This is the free KA on generators $\Sigma$
Deciding KA

- PSPACE-complete [(1 + Stock)Meyer 74]
  - automata-based decision procedure
  - nondeterministically guess a string in $L(M_1) \oplus L(M_2)$, simulate the two automata
  - convert to deterministic using Savitch’s theorem
  - inefficient—$\Omega(n^2)$ space, exponential time best-case

- coalgebraic decision procedures [Silva 10, Bonchi & Pous 12]
  - bisimulation-based
  - uses Brzozowski/Antimirov derivatives
  - Hopcroft–Karp union-find data structure, up-to techniques
  - implementation in OCaml
  - linear space, practical
Kleene Algebra with Tests (KAT)

\[(K, B, +, \cdot, *, -, 0, 1), \quad B \subseteq K\]

\begin{itemize}
  \item \((K, +, \cdot, *, 0, 1)\) is a Kleene algebra
  \item \((B, +, \cdot, -, 0, 1)\) is a Boolean algebra
  \item \((B, +, \cdot, 0, 1)\) is a subalgebra of \((K, +, \cdot, 0, 1)\)
\end{itemize}

\begin{itemize}
  \item \(p, q, r, \ldots\) range over \(K\)
  \item \(a, b, c, \ldots\) range over \(B\)
Modeling While Programs

\[ p; q \triangleq pq \]

\[ \text{if } b \text{ then } p \text{ else } q \triangleq bp + \overline{b}q \]

\[ \text{while } b \text{ do } p \triangleq (bp)^*\overline{b} \]
KAT Results

Deductive Completeness and Complexity
- deductively complete over language, relational, and trace models
- subsumes propositional Hoare logic (PHL)
- decidable in PSPACE

Applications
- protocol verification
- static analysis and abstract interpretation
- verification of compiler optimizations
NetKAT

- a packet $\pi$ is an assignment of constant values $n$ to fields $x$
- a packet history is a nonempty sequence of packets
  \[ \pi_1 :: \pi_2 :: \cdots :: \pi_k \]
- the head packet is $\pi_1$

NetKAT

- assignments $x \leftarrow n$
  assign constant value $n$ to field $x$ in the head packet
- tests $x = n$
  if value of field $x$ in the head packet is $n$, then pass, else drop
- dup
  duplicate the head packet
Example

\[ sw = 6 \; ; \; pt = 88 \; ; \; dest \leftarrow 10.0.0.1 \; ; \; pt \leftarrow 50 \]

“For all packets incoming on port 88 of switch 6, set the destination IP address to 10.0.0.1 and send the packet out on port 50.”
\[ x \leftarrow n; y \leftarrow m \equiv y \leftarrow m; x \leftarrow n \quad (x \neq y) \]
assignments to distinct fields may be done in either order

\[ x \leftarrow n; y = m \equiv y = m; x \leftarrow n \quad (x \neq y) \]
an assignment to a field does not affect a different field
NetKAT Axioms

\[ x \leftarrow n; y \leftarrow m \equiv y \leftarrow m; x \leftarrow n \quad (x \neq y) \]

assignments to distinct fields may be done in either order

\[ x \leftarrow n; y = m \equiv y = m; x \leftarrow n \quad (x \neq y) \]
an assignment to a field does not affect a different field

\[ x = n; \text{dup} \equiv \text{dup}; x = n \]
field values are preserved in a duplicated packet

\[ x \leftarrow n \equiv x \leftarrow n; x = n \]
an assignment causes the field to have that value

\[ x = n; x \leftarrow n \equiv x = n \]
an assignment of a value that the field already has is redundant

\[ x \leftarrow n; x \leftarrow m \equiv x \leftarrow m \]
a second assignment to the same field overrides the first

\[ x = n; x = m \equiv 0 \quad (n \neq m) \quad \text{(sum)} \quad \sum_n x = n \equiv 1 \]
every field has exactly one value
Standard Model

Standard model of NetKAT is a packet-forwarding model

\[[e] : H \rightarrow 2^H\]

where \(H = \{\text{packet histories}\}\)

\([x \leftarrow n](\pi_1 :: \sigma) \triangleq \{\pi_1[n/x] :: \sigma\}\)

\([x = n](\pi_1 :: \sigma) \triangleq \begin{cases} 
\{\pi_1 :: \sigma\} & \text{if } \pi_1(x) = n \\
\emptyset & \text{if } \pi_1(x) \neq n 
\end{cases}\)

\([\text{dup}](\pi_1 :: \sigma) \triangleq \{\pi_1 :: \pi_1 :: \sigma\}\)
\[ [p + q](\sigma) \triangleq [p](\sigma) \cup [q](\sigma) \]
\[ [p ; q](\sigma) \triangleq \bigcup_{\tau \in [p](\sigma)} [q](\tau) \]
\[ [p^*](\sigma) \triangleq \bigcup_n [p^n](\sigma) \]
\[ [1](\sigma) \triangleq [\text{pass}](\sigma) = \{\sigma\} \]
\[ [0](\sigma) \triangleq [\text{drop}](\sigma) = \emptyset \]
Example

Reachability

- Can host $A$ communicate with host $B$? Can every host communicate with every other host?
Encoding Network Topology

Modeling Links

sw = A; pt = n; sw ← B; pt ← m

- filters out all packets not located at the source end of the link
- updates switch and port fields to the location of the target end
- this captures the effect of sending the packet across the link
- network topology is expressed as a sum of link expressions
Switch Policies

Switch behavior for switch $A$ is specified by a NetKAT term

$$sw = A ; p_A$$

where $p_A$ specifies what to do with packets entering switch $A$

Example

$$pt = n ; dest = a ; dest \leftarrow b ; (pt \leftarrow m + pt \leftarrow k)$$

Incoming packets on port $n$ with destination $a \Rightarrow$ modify destination to $b$
and send out on ports $m$ and $k$

Switch policy $p_A$ is the sum of all such behaviors for $A$
Putting It Together

Let

- $t = \text{sum of all link expressions}$
- $p = \text{sum of all switch policies}$

Then

- $pt = \text{one step of the network}$
- each switch processes its packets, then sends them along links to the next switch
- $(pt)^* = \text{the multistep behavior of the network in which the single-step behavior is iterated}$
Reachability

To check if any packet can travel from \( A \) to \( B \) given the topology and the switch policies, ask whether

\[
sw = A ; t(pt)^* ; sw = B \not\equiv 0 \ (\text{drop}).
\]

- **prefix** \( sw = A \) filters out packets not at \( A \)
- **suffix** \( sw = B \) filters out packets not at \( B \)
Other Applications

- forwarding loops
- traffic isolation
- access control
- correctness of a compiler that maps a NetKAT expression to a set of individual flow tables that can be deployed on the switches
Results

Soundness and Completeness [Anderson et al. 14]

- $\vdash p = q$ if and only if $[p] = [q]$

Decision Procedure [Foster et al. 15]

- NetKAT coalgebra
- Efficient bisimulation-based decision procedure
- Implementation in OCaml
- Deployed in the Frenetic suite of network management tools
A Bisimulation-Based Algorithm

To check $e_1 = e_2$, convert to automata, check bisimilarity

- exploits a sparse matrix representation
- Hopcroft-Karp union-find data structure to represent bisimilarity classes
- BDDs to represent tests (new — based on Pous, POPL 15)
- algorithm is competitive with state of the art
A Bisimulation-Based Algorithm [Foster & al. 15]

- **Topology Zoo**
  - 261 real-world network topologies;
  - Use shortest path forwarding as network program;
  - Results:

- Graphs showing performance metrics such as connectivity, loop freedom, and translation validation as functions of policy size and time.
A Bisimulation-Based Algorithm [Foster & al. 15]

- **Topology Zoo**
  - 261 real-world network topologies;
  - Use shortest path forwarding as network program;
  - Results:

- **Stanford Campus Network**
  - Use actual router configurations
  - Results: Point to point reachability in 0.67s (vs 13s for HSA)
Probabilistic NetKAT

- How much congestion is there?
- Is the network resilient under failure?
- Reducing costs without compromising reliability
Probabilistic NetKAT

- How much congestion is there?
- Is the network resilient under failure?
- Reducing costs without compromising reliability

- Modular extension of NetKAT with probabilistic constructs
- Compositional semantics
- Compiler, Decision procedures, ...

Compositional quantitative reasoning $\leadsto$ fully realize the vision of SDN
10% probability of failure of the link $S_1 \rightarrow S_2$, topology $t$ encoded as:

$$t = (sw = S_1; pt = 2; (sw \leftarrow S_2; pt \leftarrow 1) \oplus_{.9 \text{ drop}})$$
$$& (sw = S_1; pt = 3; sw \leftarrow S_3; pt \leftarrow 1)$$
$$& (sw = S_2; pt = 4; sw \leftarrow S_4; pt \leftarrow 2)$$
$$& (sw = S_3; pt = 4; sw \leftarrow S_4; pt \leftarrow 3).$$
10% probability of failure of the link $S_1 \rightarrow S_2$, topology $t$ encoded as:

\[
t = (sw = S_1; pt = 2; ((sw \leftarrow S_2; pt \leftarrow 1) \oplus .9\text{ drop}))
\]

\[
& \& (sw = S_1; pt = 3; sw \leftarrow S_3; pt \leftarrow 1)
\]

\[
& \& (sw = S_2; pt = 4; sw \leftarrow S_4; pt \leftarrow 2)
\]

\[
& \& (sw = S_3; pt = 4; sw \leftarrow S_4; pt \leftarrow 3).
\]

Semantics in terms of Markov Kernels.
Conclusion

- Programming languages have a key role to play in emerging platforms for managing software-defined networks
- NetKAT is a high-level language for programming and reasoning about network behavior in the SDN paradigm
  - formal denotational semantics, complete deductive system
  - efficient bisimulation-based decision procedure
- Future work:
  - further optimizations to reduce state space
  - generating proof artifacts
  - refinement calculus
  - concurrent/distributed NetKAT
  - Many opportunities for the concurrency community!
Bridges

- Abstraction can bring new perspectives and solutions
- Transference of techniques is a two-way street
- Solid foundations are crucial for new paradigms
For papers and code, please visit:
http://frenetic-lang.org/
Thanks!